SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid *

Hong-Linh Truong[1] and Thomas Fahringer[2]

[1] Institute for Software Science, University of Vienna
truong@par.univie.ac.at
[2] Institute for Computer Science, University of Innsbruck
Thomas.Fahringer@uibk.ac.at

Abstract. This paper describes SCALEA-G, a unified monitoring and performance analysis system for the Grid. SCALEA-G is implemented as a set of grid services based on the Open Grid Services Architecture (OGSA). SCALEA-G provides an infrastructure for conducting online monitoring and performance analysis of a variety of Grid services including computational and network resources, and Grid applications. Both push and pull models are supported, providing flexible and scalable monitoring and performance analysis. Source code and dynamic instrumentation are exploited to perform profiling and monitoring of Grid applications. A novel instrumentation request language has been developed to facilitate the interaction between client and instrumentation services.

1 Introduction

Grid Monitoring is an important task that provides useful information for several purposes such as performance analysis and tuning, performance prediction, fault detection, and scheduling. Most existing Grid monitoring tools are separated into two distinct domains: *Grid infrastructure monitoring* and *Grid application monitoring*. The lack of combination of two domains in a single system has hindered the user from relating measurement metrics of various sources at different levels when conducting the monitoring and performance analysis. In addition, many existing Grid monitoring tools focus on the monitoring and analysis for Grid infrastructure; yet little effort has been done for Grid applications. To date, application performance analysis tools are mostly targeted to conventional parallel and distributed systems (e.g. clusters, SMP machines). As a result, these tools do not well address challenges in Grid environment such as scalability, diversity, dynamics and security.

To tackle above-mentioned issues, we are developing a new system named SCALEA-G. SCALEA-G is a unified system for monitoring and performance analysis in the Grid. SCALEA-G is based on the concept of Grid Monitoring Architecture (GMA) [1] and is implemented as a set of OGSA-based services [12]. SCALEA-G provides an infrastructure of OGSA-compliant grid services for online monitoring and performance analysis of a variety of Grid services including computational resources, networks, and

* This research is supported by the Austrian Science Fund as part of the Aurora Project under contract SFBF1104.

applications. Both push and pull models proposed in GMA are supported, providing a flexible and scalable way when performing the monitoring and analysis. In SCALEA-G, each kind of monitored data is described by an XML schema, allowing any client to easily access the data via XPath/XQuery. SCALEA-G supports both source code and dynamic instrumentation for profiling and monitoring events of Grid applications. A novel instrumentation request language has been devised to facilitate the interaction between client and instrumentation services. System and application specific metrics are related as close as possible in a single system, thus increasing the chance to uncover Grid performance problems.

Due to space limit, in this paper, we just describe a few selected features of SCALEA-G[3]. The rest of this report is organized as follows: Section 2 presents the architecture of SCALEA-G. In Section 3, we describe SCALEA-G sensors and Sensor Manager Service. Section 4 describes instrumentation service and instrumentation request language. We then discuss the data delivery, caching and filtering mechanism in Section 5. Security issues in SCALEA-G are outlined in Section 6. Section 7 illustrates first experiments and examples of the current prototype. We present some related work in Section 8 before going to the Conclusion and Future work in Section 9.

2 SCALEA-G Architecture

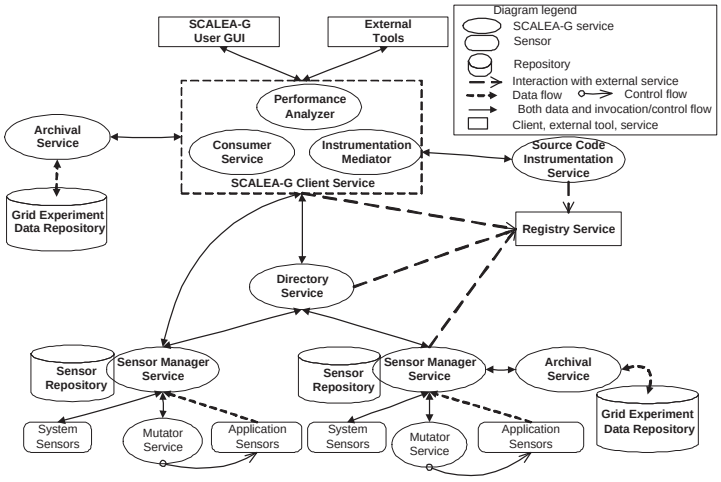

Fig. 1. High-level view of SCALEA-G Architecture.

SCALEA-G is an open architecture based on OGSA [12] combined with GMA [1]. Figure 1 depicts the architecture of SCALEA-G which consists of a set of OGSA-based services and clients. *SCALEA-G Directory Service* is used for publishing and searching information about producers and consumers that produce and consume performance data and information about types, characteristics of that data. *Archival Service* is a data repository which is used to store monitored data and performance results collected and

[3] More details can be found in ftp://ftp.vcpc.univie.ac.at/projects/aurora/reports/auroratr2003-22.ps.gz and http://www.par.univie.ac.at/project/scaleag

analyzed by other components. *Sensor Manager Service* is used to manage sensors that gather and/or measure a variety of kinds of data for monitoring and performance analysis. The instrumentation of application can be done at source code level by using *Source Code Instrumentation Service* or dynamically at the runtime through *Mutator Service*. *Client Service* provides interfaces for administrating other SCALEA-G services and accessing data in these services. In addition, it provides facilities for analyzing performance data. Any external tools can access SCALEA-G by using Client Service. *User GUI* supports the user to graphically conduct online monitoring and performance analysis; it is based on facilities provided by Client Service. SCALEA-G services register information about their service instances with a *Registry Service*.

Interactions among SCALEA-G services and clients are divided into *Grid service operation invocations* and *stream data delivery*. Grid service operation invocations are used to perform tasks which include controlling activities of services and sensors, subscribing and querying requests for performance data, registering, querying and receiving information of Directory Service. In stream data delivery, a stream channel is used to transfer data (monitored data, performance data and results) between producers (e.g. sensors, Sensor Manager Service) and consumers (e.g. Sensor Manager Service, clients). Grid service operations use transport-level and message-level security whereas data channel is secure connection; all base on Grid Security Infrastructure (GSI) [10].

In deployment of SCALEA-G, instances of sensors and Mutator Service are executed in monitored nodes. An instance of Sensor Manager Service can be deployed to manage sensors and Mutator Services in a node or a set of nodes, depending on the real system and workload. Similarly, an instance of Directory Service can manage multiple Sensor Manager Services in an administrative domain. The client discovers SCALEA-G services through registry services which can be deployed in different domains.

3 Sensors and Sensor Manager Service

SCALEA-G distinguishes two kinds of sensors: *system sensors* and *application sensors*. System sensors are used to monitor and measure the performance of Grid infrastructure. Application sensors are used to measure execution behavior of code regions and to monitor events in Grid applications. All sensors are associated with some common properties such as sensor identifier, data schema, parameters.

3.1 System Sensors and Sensor Repository

SCALEA-G provides a variety of system sensors for monitoring the most commonly needed types of performance information on the Grid investigated by GGF DAMED-WG [9] and NMWG [13].

To simplify the management and deployment of system sensors, a *sensor repository* is used to hold the information about available system sensors. Each sensor repository is managed by a Sensor Manager Service that makes sensors in the repository available for use when requested. Figure 2 presents XML schema used to express sensors in the sensor repository. The XML schema allows to specify sensor-related information such as *name* (a unique name of the sensor), *measureclass* (implementation class), *schemafile*

(XML schema of data produced by the sensor), *params* (parameters required when invoking the sensor), etc. Although not specified in the repository, by default the lifetime of a sensor instance will optionally be specified when the sensor instance is created.

```
<xsd:element name="sensorrepository"
                    type="SensorEntry"/>
<xsd:complexType name="SensorEntry">
 <xsd:sequence>
  <xsd:element name="desc"
                    type="xsd:string"/>
  <xsd:element name="measureclass"
                    type="xsd:string"/>
  <xsd:element name="schemafile"
                    type="xsd:string"/>
  <xsd:element name="params" type="Params"/>
  </xsd:sequence>
 <xsd:attribute name="name"
          type="xsd:string"/>
</xsd:complexType>
```

```
<xsd:complexType name="Params">
 <xsd:sequence>
 <xsd:element name="param"
  minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
     <xsd:attribute name="name"
                  type="xsd:string"/>
     <xsd:attribute name="desc"
                  type="xsd:string"/>
        <xsd:attribute name="dataType"
                  type="xsd:string"/>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

Fig. 2. XML schema used to describe sensors in the sensor repository.

3.2 Application Sensors

Application sensors are embedded in programs via source code instrumentation or dynamic instrumentation. Application sensors support profiling and events monitoring.

Data collected by application sensors is also described in XML format. Figure 3 shows the top-level XML schema for data provided by application sensors. The *name* tag specifies kind of sensors, either *app.event* or *app.prof*, corresponding to event or profiling data, respectively. The *experiment* tag specifies a unique identifier determining the experiment. This identifier is used to distinguish data between different experiments. The *coderegion* tag refers to information of the source of the code region (e.g. line, column). The *processingunit* tag describes the context in which the code region is executed; the context includes information about *grid site*, *computational node*, *process*, *thread*. The *events* tag specifies list of *events*, an event consists of event time, event name and a set of event attributes. The *metrics* tag specifies a list of performance metrics, each metric is represented in a tuple of name and value.

```
<xsd:element name="sensordata"  type="SensorData"/>
<xsd:complexType name="SensorData">
<xsd:sequence>
 <xsd:element name="experiment" type="xsd:string"/>
 <xsd:element name="coderegion" type="CodeRegion"/>
 <xsd:element name="events"     type="Events"/>
 <xsd:element name="metrics"    type="Metrics"/>
</xsd:sequence>
 <xsd:attribute name="name" type="xsd:NMTOKEN"/>
</xsd:complexType>
```

Fig. 3. Top-level XML schema of data provided by application sensors.

3.3 Sensor Manager Service

The main tasks of Sensor Manager Service are to control and manage activities of sensors in the sensor repository, to register information about sensors that send data to it with a directory service, to receive and buffer data sensors produce, to support data subscription and query, and to forward instrumentation request to instrumentation service.

In Sensor Manager Service, a *Data Service* receives data collected by sensor instances and delivers requested data to consumers. It implements filtering, searching, forwarding and caching data to/from various destinations/sources. In the Data Service, as shown in Figure 4, a *Data Receiver* is used to receive data from sensors and to store the received data into data buffers, and a *Data Sender* is used to deliver data to consumers. The data service uses only one connection to each consumer for delivering multiple types of subscribed data. However, an on-demand connection will be created for delivering resulting data of each query invocation and destroyed when the delivery finishes. Sensor Manager Service supports both data subscription and query. Data query requests are represented in XPath/XQuery based on XML schema published by sensors.

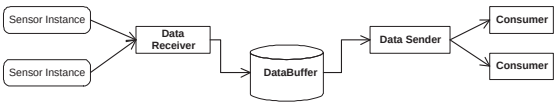

Fig. 4. Data Service in Sensor Manager Service

3.4 Interactions between Sensors and Sensor Manager Services

The interactions between sensors and Sensor Manager Services involve the exchange of three XML messages. In *initialization phase*, the sensor instance sends a *sensorinit* XML message which contains *sensor name*, an *XML schema* of data which sensor instance produces, *lifetime* and *description* information about the sensor instance to the Sensor Manager Service which then makes these information available for consumers via directory service. In *measurement phase*, the sensor instance repeatedly performs measurement, encapsulates its measurement data into a *sensordataentry* XML message, and pushes the message to the Sensor Manager Service. The measurement data is enclosed by *<![CDATA[...]]>* tag. Thus, sensors can customize the structure of their collected data. Before stopping sending collected data, the sensor instance sends a *sensorfinal* XML message to notify the Sensor Manager Service.

4 Instrumentation Service

We support two approaches: source code and dynamic instrumentation. In the first approach, we implement a Source Code Instrumentation Service (SCIS) which is based on SCALEA Instrumentation System [17]. SCIS however simply instruments input source files (for Fortran), not addressing compilation issue. Thus, the client has to compile and link the instrumented files with the measurement library containing application sensors.

In the second approach, we exploit the dynamic instrumentation mechanism based on Dyninst [6]. A *Mutator Service* is implemented as a GSI-based SOAP C++ Web service [14] that controls the instrumentation of application processes on the host where the processes are running. We develop an XML-based instrumentation request language (IRL) to allow the client to specify code regions of which performance metrics should be determined and to control the instrumentation process. The client controls the instrumentation by sending IRL requests to Mutator Services which in turn perform the instrumentation, e.g. inserting application sensors into application processes.

4.1 Instrumentation Request Language (IRL)

The IRL is provided in order to facilitate the interaction between instrumentation requester (e.g. users, tools) and instrumentation services. IRL which is an XML-based language consists of instrumentation messages: request and response. Clients send requests to Mutator Services and receive responses that describe the status of the requests.

Figure 5 outlines the XML schema of IRL. The job to be instrumented is specified by *experiment* tag. Current implementation of IRL supports four requests including *attach, getsir, instrument, finalize*:

- *attach*: requests the Mutator Service to attach the application and to prepare to perform other tasks on that application.
- *getsir*: requests the Mutator Service to return SIR (Standardized Intermediate Representation) [11] of a given application.
- *instrument*: specifies code regions (based on SIR) and performance metrics should be instrumented and measured.
- *finalize*: notifies the Mutator Service that client will not perform any request on the given application.

In responding to a request from a client, the Mutator Service will reply to the client by sending an instrumentation response which contains the name of the request, the status of the request (e.g OK, FAIL) and possibly a detailed responding information encoded in *<![CDATA[...]]>* tag.

```
<xsd:element name="irl"    type="IRL"/>
<xsd:complexType name="IRL">
 <xsd:sequence>
 <xsd:element name="experiment"
 type="Experiment" minOccurs="0" maxOccurs="1"/>
 <xsd:element name="request"  type="Request"
        minOccurs="0" maxOccurs="unbounded"/>
 <xsd:element name="response" type="Response"
       minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Request">
  <xsd:sequence>
    <xsd:element name="experiment"
                     type="Experiment" />
    <xsd:element name="task" type="Task"/>
  </xsd:sequence>
 <xsd:attribute name="name" type="xsd:NMTOKEN"/>
</xsd:complexType>
<xsd:complexType name="Experiment">
 <xsd:sequence>
  <xsd:element name="applicationName"
                    type="xsd:string"/>
  <xsd:element name="jobID"
                     type="xsd:string"/>
  <xsd:element name="experimentID"
                   type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Task">
 <xsd:sequence>
   <xsd:element name="coderegion"
                type="CodeRegion"/>
   <xsd:element name="metrics"
                type="ListString"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="CodeRegion">
   <xsd:attribute name="name"
                 type="xsd:string"/>
   <xsd:attribute name="id"
                 type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="Response">
   <xsd:sequence>
    <xsd:element name="detail"
                 type="xsd:string"/>
   </xsd:sequence>
   <xsd:attribute name="name"
               type="xsd:NMTOKEN"/>
   <xsd:attribute name="status"
               type="xsd:NMTOKEN"/>
</xsd:complexType>
<xsd:simpleType name="ListString">
 <xsd:list itemType="xsd:string"/>
</xsd:simpleType>
```

Fig. 5. XML Schema of Instrumentation Request Language.

5 Data Delivery, Caching and Filtering

Figure 6 depicts the message propagation in SCALEA-G that uses a simple tunnel protocol. In this protocol, each sensor builds its XML data messages and sends the messages to a Sensor Manager Service which stores the messages into appropriate buffers. When a client subscribes and/or queries data by invoking operations of Consumer Service, the Consumer Service calls corresponding operations of Sensor Manager Service and passes a *ResultID* to the Sensor Manager Service. The Sensor Manager Service then builds XML messages by tagging the ResultID to the data met the subscribed/queried condition and sends these messages to the Consumer Service. At the Consumer Service side, based on ResultID, the messages are filtered and forwarded to the client.

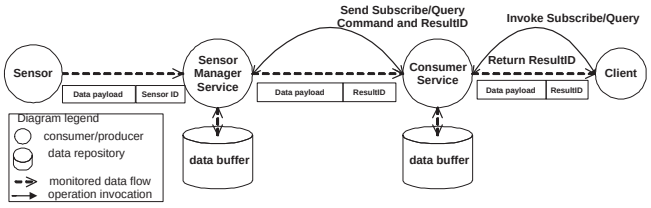

Fig. 6. Data Delivery and Caching.

Data produced by system sensors will be cached in circular bounded buffers at Sensor Manager Service. In the current implementation, for each type of system sensor, a separate data buffer is allocated for holding data produced by all instances of that type of sensor. In *push* model, any new data entry met the subscribed condition will always be sent to the subscribed consumers. In *pull* model, Sensor Manager Service only searches current available entries in the data buffer and entries met conditions of consumer query will be returned to the requested consumers. Buffering data produced by application sensors is similar to that for system sensors. However, we assume that there is only one client to perform the monitoring and analysis for each application and the size of the data buffer is unbounded.

6 Security Issues

The security in SCALEA-G is based on GSI [10] facilities provided by Globus Toolkit (GT). Each service is identified by a certificate. SCALEA-G imposes controls on clients in accessing its services and data provided by system sensors by using an Access Control List (ACL) which maps client's information to sensors the client can access. The client information obtained from client's certificate when the certificate is used in authentication will be compared with entries in the ACL in the authorization process.

The security model for Mutator Service is a simplified version of that for GT3 GRAM [10] in which Sensor Manager Service can forward instrumentation requests of clients to Mutator Service. Mutator Service runs in a none-privilege account. However, if Mutator Service is deployed to be used by multiple users, it must be able to create its instances running in the account of calling users. By doing so, the instances have permission to attach user application processes and are able to perform the dynamic instrumentation. In the case of monitoring and analyzing application, when subscribing and/or querying data provided by application sensors, client's information will be recorded. Similarly, before application sensor instances start sending data to the Sensor Manager Service, the Sensor Manager Service obtains information about the client who

executed the application. Both sources of information will be used for authorizing the client in receiving data from application sensors.

7 Experiments and Examples

We have prototyped SCALEA-G Sensor Manager Service, Directory Service, Mutator Service, a set of system and application sensors. In this section we present few experiments and examples conducted via SCALEA-G User GUI.

7.1 Administrating Sensor Manager Services and Sensors

Figure 7 presents the administration GUI used to manage activities of Sensor Manager Services. By selecting a Sensor Manager Service, a list of available sensors and a list of sensor instances managed by that Sensor Manager Service will be shown in the top-left and top-right window of Figure 7, respectively. A user (with permission) can make a request creating a new sensor instance by selecting a sensor, clicking the *Activate* button and specifying input parameters and lifetime, e.g. Figure 7 shows the dialog for setting input parameters for *path.delay.roundtrip* sensor. An existing sensor instance can be deactivated by selecting *Deactivate* button. By choosing a sensor, detailed information of that sensor (e.g. parameters, XML schema) will be shown in the two bottom windows.

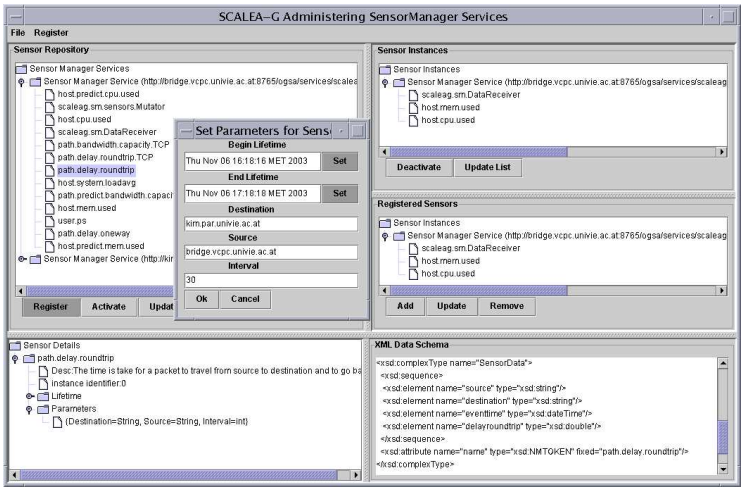

Fig. 7. SCALEA-G Administration GUI.

7.2 Dynamic Instrumentation Example

Figure 8 depicts the GUI for conducting the dynamic instrumentation in SCALEA-G. On the top-left window, the user can choose a directory service and retrieve a list of instances of Mutator Service registered to that directory service. The user can monitor processes running on compute nodes where instances of Mutator Service execute by invoking *Get/Update User Processes* operation as shown in the top-right window of

Figure 8. For a given application process, its SIR (currently only at level of program unit and function call) can be obtained via *Get SIR* operation, e.g. the SIR of $rpp3do$ process is visualized in the bottom-right window. In the bottom-left window, the user can edit IRL requests and send these requests to selected instances of Mutator Services.

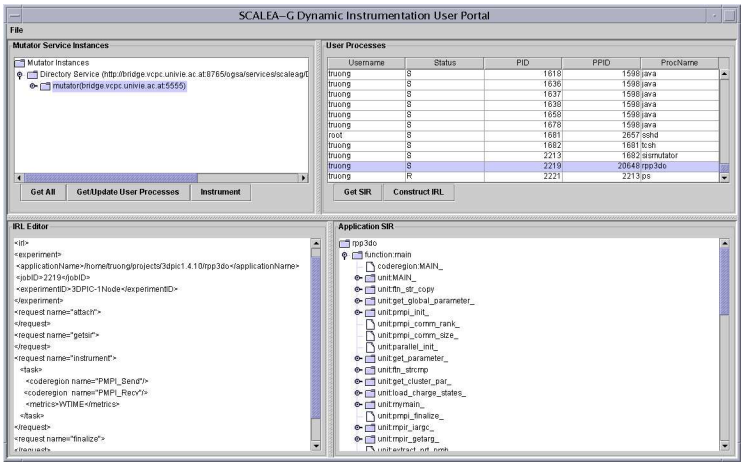

Fig. 8. SCALEA-G Dynamic Instrumentation GUI.

8 Related Work

Several existing tools are available for monitoring Grid computing resources and networks such as MDS (a.k.a GRIS) [7], NWS [18], GridRM [2], R-GMA [15]. However, few monitoring and performance analysis tools for Grid applications have been introduced. GRM [3] is a semi-on-line monitor that collects information about an application running in a distributed heterogeneous system. In GRM, however, the instrumentation has to be done manually. OCM-G [4] is an infrastructure for Grid application monitoring that supports dynamic instrumentation. Atop OCM-G, G-PM [5], targeted to interactive Grid application, is used to conduct the performance analysis. However, currently the instrumentation of OCM-G is limited to MPI functions. None of aforementioned systems, except MDS, is OGSA-based Grid service. Furthermore, existing tools employ a non-widely accessible representation for monitored data. SCALEA-G, in contrast, is based on OGSA and uses widely-accepted XML for representing performance data, and provides query mechanism with XPath/XQuery-based requests.

Although there are well-known tools supporting dynamic instrumentation, e.g. Paradyn [16], DPCL [8], these tools are designed for conventional parallel systems rather than Grids and they lack a widely accessible and interoperable protocol like IRL, thus hindering other services from using them to conduct the instrumentation.

9 Conclusion and Future Work

In this paper we presented the architecture of SCALEA-G, a unified monitoring and performance analysis system in the Grid, based on OGSA and GMA concept. The main

contributions of this paper center on the unique monitoring and performance analysis system based on OGSA and an instrumentation request language (IRL).

Yet, there are many rooms for improving the system. The set of sensors will be extended, enabling to monitor more resources and services, and providing more diverse kinds of data. In addition, the sensor will be extended to support monitoring based on resource model and rules. IRL will be extended to allow specifying more complex instrumentation requests such as events, deactivating and removing instrumentation.

References

1. B. Tierney et. al. A Grid Monitoring Architecture. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf.
2. Mark Baker and Garry Smith. GridRM: A resource monitoring architecture for the Grid. *LNCS*, 2536:268–??, 2002.
3. Zoltan Balaton, Peter Kacsuk, Norbert Podhorszki, and Ferenc Vajda. From Cluster Monitoring to Grid Monitoring Based on GRM. In *Proceedings. 7th EuroPar'2001 Parallel Processings*, pages 874–881, Manchester, UK, 2001.
4. Bartosz Balis, Marian Bubak, Włodzimierz Funika, Tomasz Szepieniec, and Roland Wismüller. An infrastructure for Grid application monitoring. *LNCS*, 2474:41–??, 2002.
5. Marian Bubak, Włodzimierz Funika, and Roland Wismüller. The CrossGrid performance analysis tool for interactive Grid applications. *LNCS*, 2474:50–??, 2002.
6. Bryan Buck and Jeffrey K. Hollingsworth. An API for Runtime Code Patching. *The International Journal of High Performance Computing Applications*, 14(4):317–329, 2000.
7. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*. IEEE Press, August 2001.
8. L. DeRose, T. Hoover Jr., and J. Hollingsworth. The dynamic probe class library: An infrastucture for developing instrumentation for performance tools. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS-01)*, Los Alamitos, CA, April 23–27 2001. IEEE Computer Society.
9. Discovery and Monitoring Event Description (DAMED) Working Group. http://www-didc.lbl.gov/damed/.
10. Von Welch et all. Security for Grid Services. In *Proceedings of 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03)*, pages 48–57, Seattle, Washington, June 22 - 24 2003.
11. T. Fahringer, M. Gerndt, Bernd Mohr, Martin Schulz, Clovis Seragiotto, and Hong-Linh Truong. Standardized Intermediate Representation for Fortran, Java, C and C++ Programs. APART Working group (http://www.kfa-juelich.de/apart/), Work in progress, June 2003.
12. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *IEEE Computer*, pages 37–46, June 2002.
13. GGF Network Measurements Working Group. http://forge.gridforum.org/projects/nm-wg/.
14. gSOAP: C/C++ Web Services and Clients. http://www.cs.fsu.edu/~engelen/soap.html.
15. R-GMA: Relational Grid Monitoring Architecture. http://www.r-gma.org.
16. Paradyn Parallel Performance Tools. http://www.cs.wisc.edu/paradyn/.
17. Hong-Linh Truong and Thomas Fahringer. SCALEA: A Performance Analysis Tool for Parallel Programs. *Concurrency and Computation: Practice and Experience*, 15(11-12):1001–1025, 2003.
18. R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems*, 15:757–768, 1999.